September 29, 1964

From: Stephen J. Garland, Dartmouth College, Hanover, N.H.

To: Potential ALGOL Users

(Revised by D. W. Scott, March 3, 1965)

A working ALGOL Compiler is now incorporated within the Time-Sharing System. At present, the following restrictions are made (most of these will be eliminated once the compiler is completed):

1. Own variables are not permitted yet, though all other block declarations are permissable; local variables are truly local and may not have the same value on re-entry to a block.

2. Dynamic array declarations are not permitted.

3. Unsigned integers may not be used as labels.

Apart from these restrictions, the full generality of ALGOL is permitted; e.g., if you like you may write for statements with arrays for running variables, recursive switches, switches whose elements are designational expressions, etc., to your heart's content.

As for grubby details, the following conventions must be obeyed when writing a program:

1. Line numbers of one to five digits generally followed by blank must be typed at the beginning of each line. However, these numbers are used solely for the purposes of editing and referencing in error messages, and are not considered as part of the program.

2. The character set for ALGOL is transliterated as follows:

abcdefg....xyz	capital letters only
0123456789	digits
+,-,*,/,↑,\	arithmetic operations (\ stands for ÷ and you will find it over the L, using the shift)
=, =/, <,<=,>,>=	relationals (/= may be used in place of =/)
true, false	logical constants
and, or, equiv, imply, not	logical operations
goto, if, then, else, for, do	sequential operators
step, until, while, comment	separators (the $ stands for the symbol $_{10}$)

, . : := ; $ begin, end, (,), [,], "	brackets (the " is used as both the left and right string quote. [and] are found above K and M, using shift.)
own, boolean, integer, real array, switch, procedure data	declarators
string, label, value	specificator (string is not implimented.)

ALGOL words in the above list are recognized by the compiler by the fact that they are spelled without spaces and are both preceeded and succeeded by a non-alphanumeric character. Note that go to will not be recognized. ALL words are made of capital letters, not underlined.

3. Identifiers may be any combination of letters and digits starting with a letter and less than thirty characters in length. Imbedded spaces are ignored (e.g., "next number in list" is a legal indentifier and is identical with "nextnumberinlist"). However, care must be taken to insure that none of the above ALGOL words are set off by spaces in an identifier (i.e., "endoflist" is legal, but "end of list" is not).

4. Aside from the restrictions in (2), spaces may be used at will to improve the appearance of a program. Carriage returns are likewise ignored by the compiler (though they do act as spaces). Any number of statements (that will fit) may be typed as one line, and one statement may be broken up onto several lines.

5. Numerical constants should be expressed with a decimal point if they are used as type real* and without if not; up to 9 significant digits are allowed.

Input-output at present is handled by the following set of procedures: For input, a new type declaration has been added to ALGOL; namely, data. The format is as follows:

data block:= 376, -96.1, 14$-7, .000003, true, false;

i.e., the word data followed by an identifier, the := sign, and a list of numbers. In Backus normal form,

<data list> ::= <number> | <data list>, <number>
<data declaration> ::= data <identifier> := <data list>

---

* In a procedure call, real numerical constants must have a decimal point or a dollar sign.

Data declarations are local to the block in which they occur, though they do not have to be given in the block heading and the appearance of such a declarations within a compound statement which contains no other declarations does not cause that compound statement to become a block. I.e., data declarations operate similar to the declarations of labels by the sign: .

The data type declaration is utilized by the procedure "readata", which is a procedure of an arbitrary number of arguments, the first of which must be a data name and the rest of which may be simple or subscripted variables. The action of the procedure readata is to assign to each variable occurring as a parameter the next number in the data block; successive calls of the procedure do not reset the pointer in the block, and the program will be terminated when any data block runs out of data. Care must be exercised by the programmer to be sure that the types of the parameters in the readata list match the types of the numbers in the data list as the compiler does not check for mismatches. It is suggested, therefore, that real, integer, and boolean data be kept in separate blocks.

Output is accomplished through the procedure "print". Again, "print" may have any number of parameters. The action of print is to tab first to the beginning of the next field of 15 characters (with no tab occurring if the type bar is at the beginning of a field) and then to print the number ala BASIC. Logical values are also printed. The ")" terminating the procedure call generates a carriage return. Strings may be printed by enclosing them in quotes and using the resulting string as a parameter. Print will check to make sure the string will fit on the current line, and if not, it will generate a carriage return. Strings being inputted through the teletypes may occupy more than one line (provided when they are printed they will fit on one line), with every space, except the first one, following line numbers of successive lines being counted as part of the string.

Tab and carriage return suppression is possible. To suppress a tab before printing, insert an empty string "" as a parameter before the number; to suppress a carriage return, a "" should be the last parameter in the list. The following are examples of print statements, and with a little fiddling at the teletype, their mysteries may become transparent:

```
print(x, a[i,], 3*sin(z), 0<i or j=17):

print("anwer =", "", x):          (tab suppressed)

for i := 1 step 1 until 10 do print("" ,a[i], "" ):

print("Now is the time for all good men to come
       to the aid of their party"):
```

The following procedures are also incorporated in the compiler:

sin		As in the ALGOL report
cos		
arctan		
ln	(natural log)	
exp		
sqrt		
abs		
sign		
entier	(integer part of)	

plus "random" which is a no-argument real procedure. It should also be pointed out that ↑ behaves as ALGOL says it should, with multiplications being performed whenever possible.

Since it is beyond all expectations that everyone will always write perfect ALGOL programs, the compiler has been equipped with a set of (sometimes) informative error messages. These messages are subject to the following interpretations:

1. "storage exhausted" -- The compiler allows 5000 words for program and array storage, with this message resulting when this area is exceeded.

2. "identifier too long" -- A maximum of thirty letters and digits is allowed in one identifier, not counting imbedded spaces.

3. "too many symbols" -- The compiler has room for approximately 170 identifiers each of which is three characters long or less. Longer identifiers decrease the capacity accordingly. Various alternatives, short of giving up, are possible: you may try eliminating some variables through the use of arrays, or you may break the program up into blocks and use the same identifier with different local connotations. This error message may also occur when too many symbols are defined at once in the same block, in which case more local variables should be used.

4. "expression too complex" -- The nesting of parentheses, brackets, and other separators has become too deep for the poor compiler to keep track of. Stop trying to type your whole program on one line and give the compiler a chance.

5. "missing operand or delimiter" -- The compiler has found two adjacent expressions, which seems to indicate that you have left out a symbol. Good bets for missing symbols are * and;.

6. "too many digits in constant" -- Only nine-significant digits are allowed.

7. "illegal constant format" -- You either have two decimal points in one constant or an illegal character following the $ sign.

8. "exponent of constant too large" -- The maximum exponent allowable in the GE-235 is 75. (In normalized form)

9. "too many constants" -- 64 distinct source program constants are permitted, not counting those which appear in array declarations and data declarations.

10. "illegal symbol after expression" -- Again a symbol like; is probably missing.

11. "illegal symbol sequence" -- This time you have probably left out a variable between two operation symbols.

12. "two nots" -- The construction not not p is not legal ALGOL.

13. "two relations together" -- Neither is the construction x=y<z.

14. "array not subscripted" -- A variable declared as an array does not have a subscript expression.

15. "illegal left part variable" -- A constant or an expression occurs to the left of an := sign.

16. "illegal subscript" -- Arrays may not have boolean subscripts.

17. "error -- suspect missing]" -- This is the best hunch, though you may be missing some other separator.

18. "incorrect number of subscripts" -- Check the declaration of the array once more (or once procedures are implemented, make sure all formal parameters which are arrays occur with the same number of subscripts each time).

19. "error -- suspect missing "then" " -- Again the best hunch, though a missing (or [may also be the cause.

20. "non-boolean expression following "if" " -- Just that.

21. "messy conditional" -- You did something wrong, perhaps like combining a conditional statement and a conditional expression in the same conditional. Or perhaps it was something less sophisticated like a wrong parenthesis count. At any rate check the coding around the conditional carefully.

22. "illegal label" -- You are attempting to use a variable or some other identifier illegally as a label.

23. "error -suspect missing)" -- Same as for].

24. "data block name missing" -- The first parameter in a readata call was not the name of a data block.

25. "error in procedure call" -- At present, this means you have the wrong number of parameters in a procedure call, or that your separator count within a parameter is off.

26. "trouble" -- This message will be improved, but at present it covers the following multitude of sins: You may be missing a [in a left part variable, or you may be missing some other separator such as an <u>if</u>, an := sign, etc. <u>Or</u> somewhere in the program you have some extra variables lying around (for instance, a BASIC type print or data statement will generate this error since "print x" and "data 15" look like identifiers to the compiler). If this latter is the case, the line number printed out will correspond to the line containing the <u>end</u> of the block in which the error occurs.

27. "illegal declaration" -- Check the format of your declarations against the ALGOL report. Specific errors in declarations are covered by some of the following messages.

28. "symbol already defined" -- Either a symbol appears in two declarations in the same block, or upon exit from a block, it is discovered that a label used within that block but not local to it has still another significance in the outer block.

29. "bound pair error -- too many colons" -- Check for missing commas in an array declaration.

30. "no colon in bound pair" -- Try putting one in.

31. "upper bound less than lower bound" -- You did something like array a [10:1];

32. "compiler error" -- You should never get this one, but if you do bring it to the attention of the authorities as it most likely indicates a machine malfunction, and only very improbably a compiler error.

33. "not in" -- Short and sweet for "wait until we get it written"; i.e., you are attempting to use a part of ALGOL which is not yet implemented.

34. "undefined label in program" -- This should be interpreted loosely to mean that somewhere in the program there is an undefined label, switch, or data name.

35. "spurious quote" -- strings may appear only as parameters to procedure print. All other occurrences generate this message. May also be caused by missing commas or parenthesis.

36. "program incomplete" -- Not enough ends's are in the program.

37. "error in for statement" -- Again the separator count is off. Check the whole statement carefully.

The following messages apply at run time:

38. " x↑y" -- The arguments x and y as printed result in an undefined result.

39. "ln of x" -- x as printed is either 0 or negative.

40. "sqrt of x" -- x as printed is negative.

41. "subscript out of bounds" -- A subscript does not fall within its declared bounds.

42. "integer too large" -- In trying to compute a subscript, the computer found an integer greater than 2↑30, which must surely have been out of bounds.

43. "overflow" -- A floating point number larger than 2↑(2↑8) has exceeded the capacity of the machine.

Most error messages are followed by "at line no. xxx" or "near line no. xxx". These messages tend to localize the error, though in the "near" variety, the error may have occurred in a previous line with the compiler picking it up only in the line given. One thing to watch for is that the compiler will not recognize an end immediately followed by a carriage return until after it has recorded the next line number. At run time, the line number given is the last one that the flow of the program has encountered; i.e., a goto a label does not cause the line number to be updated to the line containing the label.

At present, all error messages are terminal and the program must be corrected and restarted. In the near future, the compiler will be equipped to recover from most errors in order to look for others. Any complaints as to mis- or un-informative error messages should be brought to the attention of those in command, as it is extremely difficult to anticipate all the mistakes programmers may make, so that the job of providing intelligent error messages is a never-ending task.

And for the curious.....

Some types of programs will tend to run slowly in ALGOL due to the generality allowed. The outstanding example occurs in for statements, where the evaluation of everything in sight is dynamic. In most cases, efficient coding is produced where efficient coding is possible, yet the following pointer is provided for speed demons: the while type element is faster than the step until. In programs involving long arithmetic statements, very clean coding is produced, so that the lack of speed in for statements is somewhat offset.

COMING ATTRACTIONS -- an intermediate input procedure which will make possible game playing routines and other programs involving human interaction; an elapsed time procedure for timing loops and other computations; more and better error messages; a general formatted output procedure; and other assorted goodies. If you have any suggestions concerning features you would like to see implemented in a compiler, let some one know; it's most likely impossible, impractical, or a darn nuisance, but it doesn't hurt to try.